Differentially Private Histograms for Range-Sum Queries: A Modular Approach

Georgios Kellaris[1] Stavros Papadopoulos[2] Dimitris Papadias[1]

[1]HKUST [2]Intel Labs & MIT
{gkellaris, dimitris}@cse.ust.hk stavrosp@csail.mit.edu

ABSTRACT

We focus on the problem of differentially private histogram publication, for range-sum query answering. Specifically, we derive a histogram from a given dataset, such that (i) it satisfies ϵ-differential privacy, and (ii) it achieves high utility for queries that request the sum of contiguous histogram bins. Existing schemes are distinguished into two categories: fast but oblivious to utility optimizations that exploit the data characteristics, and data-aware but slow. We are the first to address this problem with emphasis on *both* efficiency and utility. Towards this goal, we formulate a principled approach, which defines a small set of simple *modules*, based on which we can devise a variety of more complex schemes. We first express the state-of-the-art methods in terms of these modules, which allows us to identify the performance bottlenecks. Next, we design novel efficient and effective schemes based on non-trivial module combinations. We experimentally evaluate all mechanisms on three real datasets with diverse characteristics, and demonstrate the benefits of our proposals over previous work.

1. INTRODUCTION

A *histogram* computed on a dataset D is a vector of counts, such that each record in D affects at most a single histogram element, called *bin*. The histogram constitutes one of the most basic statistical tools for describing the dataset distribution. A *range-sum* query over a histogram returns the sum of values of a set of *contiguous* bins. Our goal is to publish a histogram on D that satisfies *ϵ-differential privacy* [6]. This paradigm entails perturbing the bins prior to their publication, so that each individual record in D is protected. We aim at minimizing the total error incurred by the perturbation when answering *arbitrary* (i.e., not known a priori) range-sum queries. For example, consider a database D with medical records, and a histogram on D where each bin contains the number of patients of a certain age. Assuming bins sorted on age, a range-sum query over this histogram returns the number of patients in the range. In this scenario, it is important that the published histogram does not violate the privacy of any individual patient; at the same time the range-sum results should be accurate.

Differentially private histogram publication has been studied extensively. The existing schemes can be divided into two categories. "Data-aware" methods exploit the underlying dataset distribution [2, 20, 22, 13]. They *smooth* the histogram prior to its perturbation by grouping *similar* bins and replacing their values with the group *average*. This yields reduced perturbation per bin. However, these mechanisms exhibit superquadratic time complexities, which may be prohibitive in time-critical applications. "Data-oblivious" methods build a perturbed *aggregate tree* on top of the histogram, and answer range-sum queries by summing a small number of tree nodes, instead of numerous individual bins falling in the range [4, 10, 17, 19]. Such approaches are very efficient, running in time linear to the number of histogram bins, but may yield low utility for some practical datasets.

To the best of our knowledge, this is the first work that aims at efficiency, without compromising utility. Towards this goal, we address the problem in a principled, *modular* approach. Specifically, we first identify three building blocks, which we call *modules*, namely *Smoothing*, *Hierarchy Level*, and *Fixed Queries*; the first is inspired by the data-aware techniques, the second works on a single level in the aggregate tree, and the third is based on techniques such as the matrix mechanism [14], which has been applied to increase utility for fixed query workloads [13]. We then formulate a *scheme* as a combination of these modules, integrated with certain components, called *connectors*. The latter do not affect privacy, but serve to properly format the inputs of the modules. Subsequently, we express the existing state-of-the-art methods in our modular framework, and discover opportunities for optimization. Finally, we devise novel efficient and effective schemes by composing the modules non-trivially. Concretely, our contributions are:

- We introduce a modular framework on differentially private histograms for range-sum queries. Our approach offers multiple benefits: (i) using a small set of simple modules and connectors, we can analyze all existing methods, and devise novel schemes with variable efficiency and utility, (ii) given the privacy level of each module, we can easily derive the privacy of arbitrarily complex schemes, and (iii) each module can be optimized separately; furthermore, potential future optimizations can be incorporated to an existing scheme with minimal effort.

- We analyze an important submodule of the Smoothing module, namely the grouping method, which essentially solves an optimization problem. We point out the two objective functions most heavily used in the literature, propose optimizations and evaluate their effect on utility.
- We design novel schemes based on the defined modules, including the first mechanism that seamlessly combines the data-aware and -oblivious methodologies. In addition, we efficiently adapt schemes for fixed query workloads (e.g., the matrix mechanism) to arbitrary range-sums, via a simple but powerful technique based on *prefix sums* [11].
- We provide a thorough experimental evaluation that compares the best existing methods with our new solutions, testing over three real datasets with different characteristics. We exhibit that there is a trade-off between algorithmic efficiency and utility across the various approaches.

The remainder of the paper is organized as follows. Section 2 includes preliminary information and surveys the related work. Section 3 introduces our modular framework. Section 4 investigates grouping in depth. Section 5 presents our proposed mechanisms. Section 6 experimentally evaluates all schemes, whereas Section 7 concludes our work.

2. BACKGROUND

Section 2.1 formulates our setting, and includes the necessary primitives on differential privacy. Section 2.2 surveys differentially private histogram publication.

2.1 Setting and Primitives

Let $\mathcal{D}$ be a collection of datasets. We define a family of functions $\mathcal{F} = \{F_j : \mathcal{D} \to \mathcal{H}\}$, such that for all j and all $D \in \mathcal{D}$, $F_j(D) = \mathbf{h} \in \mathcal{H}$ is an (ordered) vector called *histogram*. An element of $\mathbf{h}$ is termed *bin* and consists of a value and a label, where $\mathbf{h}[i]$ represents the i^{th} bin value of $\mathbf{h}$. All histograms have the property that any record in D increments *at most a single* $\mathbf{h}[i]$ by 1. Finally, we call F_j a *histogram algorithm*. For instance, let $D \in \mathcal{D}$ be a dataset of medical records. Then, $F_1 \in \mathcal{F}$ may produce histogram $\mathbf{h}_1$ such that $\mathbf{h}_1[i]$ is the number of patients in D having age i, and $F_2 \in \mathcal{F}$ may produce histogram $\mathbf{h}_2$ such that $\mathbf{h}_2[i]$ is the number of patients in hospital with id i. Observe that, the presence of a patient in D increments at most one bin by 1 in both histograms.

Our goal is to publish a n-element histogram $\mathbf{h}$ produced by some *fixed* algorithm F on a $D \in \mathcal{D}$, while satisfying ϵ-*differential privacy* and allowing *arbitrary range-sum queries* on its bins with high utility. Specifically, we define a range-sum query as a range of bins $[i_l, i_u]$, $1 \leq i_l \leq i_u \leq n$, which returns the sum $\sum_{i=i_l}^{i_u} \mathbf{h}[i]$. In our example above, a range-sum query on $\mathbf{h}_1$ could be $[10, 20]$, asking for the number of patients between 10 and 20 years old. We assume that the range queries are *not known* prior to the publication of the histogram.

To achieve ϵ-differential privacy, we apply a mechanism M on the histogram, which perturbs it in a way that satisfies the following definition, adapted from [5].

Definition 1. *A mechanism $M : \mathcal{H} \to \hat{\mathcal{H}}$ satisfies* **ϵ-differential privacy** *for a histogram algorithm $F \in \mathcal{F}$, if for all sets $\hat{H} \subseteq \hat{\mathcal{H}}$, and every pair $D, D' \in \mathcal{D}$ where D' is obtained from D by removing a record (D, D' are called neighboring), it holds that*

$$\Pr[M(F(D)) \in \hat{H}] \leq e^{\epsilon} \cdot \Pr[M(F(D')) \in \hat{H}]$$

Intuitively, ϵ-differential privacy guarantees that the perturbed histogram $\hat{H}$ will be the same with high probability (tunable by ϵ), regardless of whether a patient agrees to participate in the publication or not. Equivalently, the sensitive information of any patient cannot be inferred from the published data.

Definition 2. *The* **sensitivity** *of any histogram algorithm $F \in \mathcal{F}$ is $\Delta(F) = \max_{D, D' \in \mathcal{D}} \|F(D) - F(D')\| = 1$ for all neighboring $D, D' \in \mathcal{D}$.*

In other words, the sensitivity of F represents how much the histogram $F(D)$ changes when a record is deleted from D. Since any record contributes 1 to at most a single bin, the sensitivity is 1 for any histogram algorithm $F \in \mathcal{F}$.

The most basic technique to achieve ϵ-differential privacy is to add Laplace noise to the histogram bins using the Laplace Perturbation Algorithm (LPA [7, 6]). Let $Lap(\lambda)$ be a random variable drawn from a Laplace distribution with mean zero and scale parameter λ. LPA achieves ϵ-differential privacy through the mechanism outlined in the following theorem, adapted from [7].

Theorem 1. *Let $F \in \mathcal{F}$ and define $\mathbf{h} \stackrel{\text{def}}{=} F(D)$. A mechanism $\mathcal{M}$ that adds independently generated noise from a zero-mean Laplace distribution with scale parameter $\lambda = \Delta(F)/\epsilon = 1/\epsilon$ to each of the values of $\mathbf{h}$, i.e., which produces transcript $\hat{\mathbf{h}} = \mathbf{h} + \langle Lap(1/\epsilon) \rangle^n$, enjoys ϵ-differential privacy.*

With LPA, a range-sum query $[i_l, i_u]$ is processed on the noisy $\hat{\mathbf{h}}$ and returns $\sum_{i=i_l}^{i_u} \hat{\mathbf{h}}[i]$. The Laplace noise injected in each bin introduces error, which is aggregated when the noisy bin values are added. For large ranges, this error may completely destroy the utility of the answer. Numerous works (overviewed in Section 2.2) introduce alternative mechanisms for improving the utility of the output histograms in the case of range-sum queries.

Finally, we include a *composition* theorem (adapted from [16]) that is useful for our proofs. It concerns executions of multiple differentially private mechanisms on non-disjoint and disjoint inputs.

Theorem 2. *Let $M_1, \ldots, M_r$ be mechanisms, such that each M_i provides ϵ_i-differential privacy. Let $\mathbf{h}_1, \ldots, \mathbf{h_r} \in \mathcal{H}$ be histograms created on pairwise non-disjoint (resp. disjoint) datasets $D_1, \ldots, D_r$, respectively. Let M be another mechanism that executes $M_1(\mathbf{h}_1), \ldots, M_r(\mathbf{h}_r)$ using independent randomness for each M_i, and returns their outputs. Then, M satisfies $\left(\sum_{i=1}^{r} \epsilon_i\right)$-differential privacy (resp. $(\max_{i=1}^{r} \epsilon_i)$-differential privacy).*

The above theorem allows us to view ϵ as a *privacy budget* that is distributed among the r mechanisms. Moreover, note that the theorem holds even when M_i receives as input the private outputs of $M_1, \ldots, M_{i-1}$ [16].

2.2 Differentially Private Histograms

Existing literature on differentially private histograms for range-sum queries aims at improving upon LPA in terms of utility. We divide the approaches into two categories; *data-aware* that utilize *smoothing*, and *data-oblivious* that rely on *hierarchical* tree structures.

Data-aware methods. These approaches first smooth the histogram, typically either by grouping similar bin values and substituting them with their average, or by performing a smoothing filter such as the Discrete Fourier Transform (DFT). Subsequently, they apply Laplace noise similar to LPA to the averages or the DFT coefficients. Range-sum queries are processed by summing the histogram bin values in the query range. Smoothing reduces the sensitivity and, hence, the injected Laplace noise, but adds approximation error. Consequently, smoothing methods are effective if the Laplace noise error reduction exceeds the smoothing approximation error. The bin grouping algorithm assigns scores to a set of potential grouping strategies, and selects the one with the minimum score, in a manner that does not compromise differential privacy. Existing approaches differ in the set of examined strategies, the scoring function, and the selection process.

The SF algorithm [20] follows the grouping and averaging paradigm. Specifically, given as input a *fixed* parameter k and privacy budgets ϵ, ϵ', SF initially finds a set of k groups of *contiguous* bins through an ϵ'-differentially private process. Subsequently, it smooths the bin values based on the grouping, and adds Laplace noise generating $(\epsilon - \epsilon')$-differentially private histogram. Due to linear composition (Theorem 2), the SF mechanism achieves ϵ-differential privacy. The grouping sub mechanism of SF operates on the original histogram and determines the k groups such that the estimated *squared* error is minimized. This error is expressed as the sum of (i) the squared approximation error due to smoothing, and (ii) the squared error from injecting Laplace noise with scale $1/(\epsilon - \epsilon')$ prior to publication. It then applies the exponential mechanism [15] in order to alter the group borders and achieve ϵ'-differential privacy. Note that, due to this step, the total error of SF eventually deviates from the actual minimum. The grouping submodule of SF runs in $O(n^2)$.

Acs et al. [2] present two mechanisms, EFPA and P-HP. EFPA is an improvement of [18], which smooths the histogram using a subset of its DFT coefficients perturbed with Laplace noise, while guaranteeing that the output histogram satisfies ϵ-differential privacy. P-HP is a grouping and averaging method that improves SF [20]. In particular, instead of receiving the number of groups k as input, it discovers the optimal value of k on-the-fly. Contrary to SF, it utilizes an *absolute* error metric. The grouping algorithm of P-HP runs also in $O(n^2)$, but similarly to SF does not examine all possible groups. P-HP is shown to outperform both EFPA and SF in terms of utility [2].

Motivated by [12] (for a different setting), AHP [22] first applies LPA to the histogram with scale $1/\epsilon'$, and *sorts* the resulting bins in descending or ascending order. Subsequently, it executes a grouping and averaging technique that is different from SF and P-HP. Specifically, it operates on already ϵ'-differentially private data and, hence, does not need to apply the exponential mechanism. Moreover, it finds the grouping that minimizes the *squared* error metric expressed as a function of the noisy data, rather than the original histogram (and, thus, similar to [20, 2], it does not guarantee the actual minimum error). Note that the ordering attempts to minimize the approximation error, since it results in groups with more uniform bin values. The authors present two algorithms; one that evaluates all possible groups and runs in $O(n^3)$ time, and a greedy one that considers only a subset of the possible options and runs in $O(n^2)$. They conducted experiments using the latter, and demonstrated that AHP offers better utility than P-HP.

DAWA [13] comprises of two stages. The first stage executes a smoothing technique, while the second an optimized version of the matrix mechanism [14]. Its grouping and averaging submodule invests ϵ' budget to reduce the *absolute* error metric similar to [2]. However, instead of executing the exponential mechanism, it adds noise to the costs of the groups used in the selection process on-the-fly. The authors present two instantiations; the first evaluates all possible groupings and runs in $O(n^2 \log n)$ time, whereas the second considers only a subset and runs in $O(n \log^2 n)$. The output of the smoothing procedure is fed to the matrix mechanism. The latter belongs to a category of schemes [14, 21, 9] that take as input a set of *pre-defined* range-sum queries, and assign more privacy budget to the bins affecting numerous queries. DAWA can be adapted to our setting of *arbitrary* queries in two ways; either by completely ignoring the second stage, resulting in time complexity $O(n^2 \log n)$ (or $O(n \log^2 n)$ in the approximate version), or by feeding all the $O(n^2)$ possible queries to the input of the matrix mechanism, yielding time complexity $O(n^3 \log n)$.

Data-oblivious methods. These schemes build an aggregate tree on the original histogram; each bin value is a leaf, and each internal node represents the sum of the leaves in its subtree. In order to achieve ϵ-differential privacy, they add Laplace noise to each node, which is proportional to the tree height (since each bin value is incorporated in all the sums along its path to the root). A range-sum query is processed by identifying the maximal subtrees that exactly cover the range, and summing the values stored in their roots. Compared to LPA, the hierarchical methods essentially increase the sensitivity from 1 to $\log n$, but sum fewer noisy values when processing the range-sum, reducing the aggregate error. For a range-sum covering m bins, these methods induce $O(\log m \log n)$ error, as opposed to LPA that inflicts $O(m)$ error. Therefore, the hierarchical methods exhibit benefits for large ranges. Moreover, their time complexity is $O(n)$.

Hay et al. [10] build a binary aggregate tree and inject Laplace noise uniformly across all nodes. In addition to constructing the final range-sum from the roots of the maximal subtrees that cover the range, they also explore other node combinations. Independently from [10], Privelet [19] builds a Haar wavelet tree and adds Laplace noise, achieving practically the same effect as [10]. Based on the observation that the privacy budget should not be divided equally among all levels, Cormode et al. [4] enhance [10] with a geometric budget allocation technique. Qardaji et al. [17] survey the above approaches, concluding that the theoretical optimal fan-out of the tree is 16. They experimentally showed that [10], when combined with the budget allocation of [4] and their optimal fan-out, outperforms Privelet and SF.

Discussion. Data-oblivious methods are fast, but may have low utility for practical datasets. Data-aware schemes avoid

this by exploiting the underlying data, but they may be prohibitively slow. For example, assuming the squared error metric, the lowest time complexity achieved by any method is $O(n^3)$. Attempts to boost performance via approximation, by ignoring possible groupings, compromise utility in an unpredictable way. Moreover, the naive adaptation of DAWA to arbitrary queries, by feeding all the $O(n^2)$ possible range-sums, is impractical.

Finally, all the discussed methods involve common components. For instance, data-aware schemes only differ in their grouping technique (e.g., different error metrics in the scoring function), whereas data-oblivious methods only differ in the tree fanout and the budget allocation across the levels. These design decisions are orthogonal; e.g., we could use the tree fan-out of one method with the budget allocation policy of another. Going one step further, novel methods could combine the merits of both data-aware and -oblivious schemes.

Motivated by the above, in this work we formulate a principled approach, which defines the core privacy techniques as primitive modules. Our framework allows (i) the careful study and optimization of each individual module, (ii) the construction of efficient and effective schemes via the seamless combination of these modules, and (iii) the effortless adaptation of additional modules, such as the matrix mechanism, in our problem setting.

3. MODULAR FRAMEWORK

Section 3.1 formulates the concept of *module* along with related notions. Section 3.2 describes the module instantiations utilized to construct range-sum schemes. Section 3.3 demonstrates how the existing state-of-the-art range-sum schemes (used as competitors in our experiments) can be expressed in our modular framework.

3.1 Definitions

There are two types of building blocks in our approach: the *module* and the *connector*, formulated in the next two definitions.

DEFINITION 3. *A* **module** *is a mechanism that takes as input a sensitive histogram* $\mathbf{h} \in \mathcal{H}$ *and a vector of public parameters* $\mathbf{p}$*, and outputs a differentially private histogram* $\hat{\mathbf{h}} \in \hat{\mathcal{H}}$*. The privacy level (i.e.,* ϵ*) of the module depends on* $\mathbf{h}, \mathbf{p}$*, and its internal mechanics.*

DEFINITION 4. *A* **connector** *is an algorithm that takes as input a vector of public parameters* $\mathbf{p}$*, and either* $H \subseteq \mathcal{H}$ *(i.e., sensitive histograms) or* $\hat{H} \subseteq \hat{\mathcal{H}}$ *(i.e., differentially private histograms), and outputs another vector of parameters* $\mathbf{p}'$*, along with sets* $H' \subseteq \mathcal{H}$ *and* $\hat{H}' \subseteq \hat{\mathcal{H}}$*. It must obey two constraints: (i) it must spend* no *privacy budget, and (ii) if it takes as input some* $\mathbf{h} \in \mathcal{H}$*, all its outputs must be consumed by modules.*

Simply stated, modules are responsible for perturbing sensitive data with noise, whereas connectors *connect* modules (and optionally also other connectors). The connectors essentially format the data prior to feeding them to the modules. The public parameters facilitate determining the amount of noise added by a module. The second condition of the connectors is due to technical purposes in our proofs, which will become clear later in this section. Hereafter, we denote a module by M and a connector by C. Finally, note that a module may be further comprised of other modules and connectors, in which case we refer to it as *composite*. The motivation behind distinguishing connectors from modules is to compartmentalize the components related to privacy within the scope of a module, so that we facilitate the understanding of its privacy level and possible optimization.

DEFINITION 5. *A* **range-sum scheme** *consists of a directed acyclic graph (DAG) of modules and connectors, and a query processor. It takes as input a histogram* $\mathbf{h} \in \mathcal{H}$*, public parameters* $\mathbf{p}$*, and privacy budget* ϵ*. The DAG of modules and connectors outputs a* structure $\hat{\mathbf{S}}$ *(e.g., a histogram or tree) that satisfies* ϵ*-differential privacy, which is fed to the query processor. The latter uses the structure to answer arbitrary range-sum queries.*

Note that the above definition can capture even *iterative* schemes, such as MWEM [9], as follows. We decompose a loop into modules, and then *serialize* the loop by repeating its modules as many times as the number of loop iterations. We do not delve into more details, as we do not deal with iterative schemes in this work.

The next theorem formulates ϵ-differential privacy for a range-sum scheme. Intuitively, it states that the connectors do not affect privacy at all. The privacy level of the entire scheme depends *solely* on the modules and, thus, it suffices to analyze each module individually.

THEOREM 3. *Let a range-sum scheme comprised of modules* $M_1, M_2, \ldots, M_r$ *and connectors* $C_1, C_2, \ldots, C_l$*. Suppose that* $M_1, M_2, \ldots, M_r$ *work on sensitive inputs derived from pairwise non-disjoint (resp. disjoint) datasets, and each* M_i *satisfies* ϵ_i*-differential privacy. Then, the scheme satisfies* $\left(\sum_{i=1}^{r} \epsilon_i\right)$*-differential privacy (resp.* $(\max_{i=1}^{r} \epsilon_i)$*-differential privacy).*

PROOF. We distinguish two cases, assuming for now that the connectors take single inputs and produce single outputs: (i) A connector C takes as input a differentially private histogram $\hat{\mathbf{h}} \in \hat{\mathcal{H}}$ from a module M_i. Since C spends zero privacy budget by definition, its output will retain the privacy level of the input, independently of the computations it performs. Therefore, we can devise a module M_i' that encompasses M_i and C, and retains the ϵ_i-differential privacy of M_i. (ii) A connector C takes as input a sensitive histogram $\mathbf{h} \in \mathcal{H}$ from the scheme input. By definition, C can only produce a sensitive histogram $\mathbf{h}' \in \mathcal{H}$ as output and direct it to a module M_i. Hence, we can trivially merge C with M_i to create a module M_i' that retains the ϵ_i-differential privacy of M_i.

Replicating connectors to simulate multiple inputs and outputs, and executing the processes described in the above two cases repeatedly, from a DAG of $M_1, \ldots, M_r$ modules and $C_1, \ldots, C_l$ connectors we can derive an equivalent DAG of mechanisms $M_1', \ldots, M_r'$, where M_i' satisfies ϵ_i-differential privacy. Due to Theorem 2, the scheme satisfies $\left(\sum_{i=1}^{r} \epsilon_i\right)$-differential privacy (resp. $(\max_{i=1}^{r} \epsilon_i)$-differential privacy). □

As a final remark on privacy, recall the second constraint we imposed on the connector. If $\mathbf{h}$ were the input of a connector C whose output was not directed to a module, C could have been allowed to send $\mathbf{h}$ to the output of the range-sum scheme, violating differential privacy. The constraint prevents this case.

The benefits of modularity are threefold: (i) novel schemes with variable efficiency and utility can be developed based on a small set of simple modules and connectors, (ii) given the privacy level of each module and using Theorem 3, we can easily prove the privacy of complex schemes, and (iii) the modules can be optimized independently, and incorporate potential future improvements.

In the following, we deconstruct existing techniques into modules and connectors in order to investigate their performance bottlenecks, and identify opportunities for improvement. The internals of composite modules and schemes are illustrated using figures, depicting a module with a rectangle, a connector with a diamond, and a query processor with a parallelogram.

3.2 Atomic Modules

The three basic modules in our framework are *Smoothing*, *Hierarchy Level*, and *Fixed Queries*. These modules are composite, i.e., they consist of other modules and connectors. However, they are used as *atomic*[1] blocks when analyzing existing and novel schemes in later sections. We next explain each module in turn.

Smoothing module. This module constitutes a building block for the data-aware techniques. It imposes an order on the bins of the input histogram, groups and averages bins, applies noise, and outputs the perturbed histogram. Figure 1 depicts the internal mechanics of the Smoothing module in more detail. Its input consists of the initial histogram $\mathbf{h}$, and public parameters $\mathbf{p}$ that include a vector $\mathbf{L}$, an error metric μ (absolute or squared), and three privacy budgets $\epsilon_1, \epsilon_2, \epsilon_3$. Each element of $\mathbf{L}$ has the form $\langle g_i, v_i \rangle$, where g_i is some encoding for a group of histogram bins, and v_i quantifies the error in g_i due to the subsequent addition of noise (the value of v_i will be elaborated shortly).

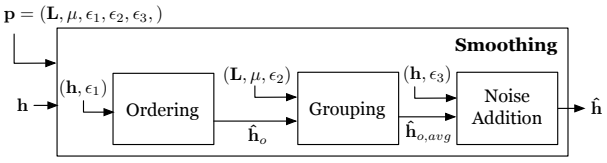


Figure 1: Smoothing module

The module consists of three submodules, called Ordering, Grouping, and Noise Addition. Ordering receives the histogram $\mathbf{h}$ and budget ϵ_1, and works as in [22]; it adds (Laplace) noise with scale $\lambda = 1/\epsilon_1$ to each bin value, and sorts them in descending order. It then forwards the noisy sorted histogram $\hat{\mathbf{h}}_o$ to the Grouping submodule.

The Grouping submodule spends budget ϵ_2 to discover the groups for its input histogram, considering $\mathbf{L}$ and μ. $\mathbf{L}$ (i) describes the *permissible groups*, and (ii) includes error values v_i that parameterize μ. A permissible group g_i can only contain *contiguous* bins, and is encoded simply by a range of elements in $\hat{\mathbf{h}}_o$, but is independent of the corresponding bin labels or values. After determining the groups, the submodule incorporates a group id into each bin label. Finally, it outputs the result, which is denoted by $\hat{\mathbf{h}}_{o,avg}$.

[1]The submodules and connectors of an atomic module are never used outside of this particular module.

The tasks performed by Grouping are elaborated further in Section 4.

Noise Addition receives the original histogram $\mathbf{h}$, budget ϵ_3, and the output $\hat{\mathbf{h}}_{o,avg}$ of the Grouping submodule. It groups the bins of $\mathbf{h}$ according to the (augmented with group ids) bin labels in $\hat{\mathbf{h}}_{o,avg}$, and averages their values. Then, it adds noise to the respective average with scale $1/(\epsilon_3 \cdot |g_i|)$. Finally, it sets the noisy average of every group g_i as the value of the bins in g_i, and outputs the noisy smoothed histogram $\hat{\mathbf{h}}$, which satisfies ϵ_3-differential privacy[2]. Note that the bin labels in $\hat{\mathbf{h}}$ incorporate the group ids of $\hat{\mathbf{h}}_{o,avg}$.

Ordering is ϵ_1-, Grouping is ϵ_2-, and Noise Addition is ϵ_3-differentially private, and they all operate on histograms derived from pairwise non-disjoint inputs. Hence, due to Theorem 2, Smoothing satisfies $(\epsilon_1 + \epsilon_2 + \epsilon_3)$-differential privacy. If we set $\epsilon_1 = 0$ ($\epsilon_2 = 0$), the Ordering (Grouping) submodule acts as a connector. Specifically, Ordering just outputs the input histogram, whereas Grouping outputs the best strategy without spending privacy budget. However, based on Definition 4, it is not permitted to simultaneously set $\epsilon_1 = 0$ and $\epsilon_2 = 0$; in that case, Ordering would forward a sensitive histogram $\mathbf{h}$ to another connector. Finally, if we set $\epsilon_3 = 0$, the Noise Addition submodule adds noise with infinite scale to each group average. Although the returned $\hat{\mathbf{h}}$ contains useless values, its bin labels incorporate the grouping information from the Grouping submodule.

Hierarchy Level. This is a typical component of the data-oblivious schemes. Recall that these methods build an aggregate tree on the original histogram. Every level of the tree can be viewed as a separate histogram. The Hierarchy Level module operates on a histogram of a specific tree level. Figure 2 illustrates its internal parts. The module receives a histogram $\mathbf{h}$, a vector $\mathbf{L}$, privacy budget ϵ, tree height t, and a tree level ℓ. It consists of two connectors (Scale Budget and Scalar Product), and a submodule Noise Addition.

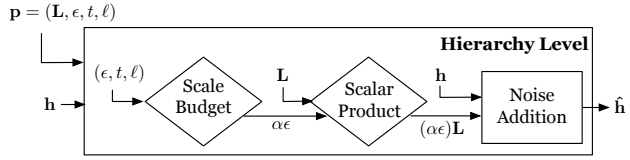


Figure 2: Hierarchy Level module

In our implementation, the Scale Budget connector allocates the privacy budget based on the tree level, using the method of [4] to maximize utility. It receives as input the triplet (ϵ, t, ℓ), and outputs $\alpha\epsilon$, i.e., it determines a parameter α that scales budget ϵ. The Scalar Product connector takes as input $\alpha\epsilon$ and public vector $\mathbf{L}$, and simply outputs their scalar product $(\alpha\epsilon)\mathbf{L}$. This essentially distributes the budget assigned for the level (potentially) non-uniformly over the bins. The output $(\alpha\epsilon)\mathbf{L}$ is forwarded to the Noise Addition submodule, which adds noise with scale $1/((\alpha\epsilon)\mathbf{L}[i])$ to the i^{th} bin of the histogram, and outputs the resulting noisy histogram $\hat{\mathbf{h}}$.

[2]Contrary to LPA, Smoothing distributes the noise *non-uniformly* over the bins of $\mathbf{h}$. This can be thought of as splitting $\mathbf{h}$ into $|G|$ *disjoint* histograms, each corresponding to a $g_i \in G$ and, due to averaging, having sensitivity $1/|g_i|$. Due to Theorem 1, injecting noise with scale $1/(\epsilon_3|g_i|)$ renders each histogram ϵ_3-differentially private. Due to Theorem 2, Smoothing is also ϵ_3-differentially private.

The $\mathbf{L}$ parameter is selected, so that the Hierarchy Level module is $(\alpha\epsilon)$-differentially private. In our schemes, we distinguish two cases: (i) $\mathbf{L} = \mathbf{1}^n$, and every bin receives the same noise with scale $1/(\alpha\epsilon)$. (ii) $\mathbf{L}[i] = 0$ for some bins, in which case the module adds noise with infinite scale. Observe that in both cases, the added noise achieves $(\alpha\epsilon)$-differential privacy.

Fixed Queries. This module is the building block of methods that target at range-sum queries known a priori. It receives as input a histogram $\mathbf{h}$, a privacy budget ϵ, and a range-sum query workload $\mathbf{W}$. It executes an off-the-shelf mechanism such as MWEM [9] or the matrix mechanism [14], and outputs the noisy histogram $\hat{\mathbf{h}}$. Figure 3 shows the Fixed Queries module, instantiated with the optimized matrix mechanism submodule of [13], used in our implementation.

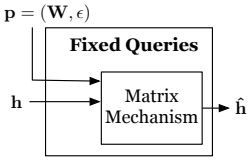


Figure 3: Fixed Queries module

3.3 Modularizing Existing Schemes

In this section we show how the existing approaches can be constructed using modules and connectors.

Smoothing scheme. All data-aware mechanisms [20, 2, 22, 13] described in Section 2.2 are captured by the scheme of Figure 4, which is a simple combination of the Smoothing module with a Query Processor. The latter receives the noisy histogram output by the Smoothing module, and replies to range-sum queries. The queries are processed by adding the bins falling in the query range.

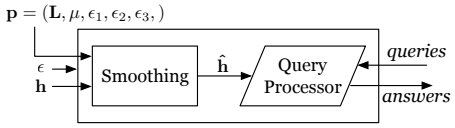


Figure 4: Smoothing scheme

Depending on the choice of public parameters $\mathbf{p}$, we can have the following alternative scheme instantiations:

- *With/without ordering:* We can deactivate (activate) the Ordering submodule by setting the value of ϵ_1 to 0 (> 0). For instance, if we set $(\epsilon_1, \epsilon_2, \epsilon_3) = (\epsilon/2, 0, \epsilon/2)$ we reproduce AHP [22]. Note that $\epsilon_2 = 0$ because the Grouping submodule operates directly on noisy data and does not need to inject extra noise (i.e., it acts as a connector). On the other hand, if we set $(\epsilon_1, \epsilon_2, \epsilon_3) = (0, \epsilon/4, 3\epsilon/4)$, we reproduce the smoothing scheme of DAWA [13]. Observe that either case results in an $(\epsilon_1+\epsilon_2+\epsilon_3 = \epsilon)$-differentially private Smoothing module. Since this is the only module in the scheme, the Smoothing scheme is also ϵ-differentially private.

- *Exact/approximate grouping:* The Grouping submodule can be implemented either as an exact or an approximate algorithm. In the first case, public parameter $\mathbf{L}$ includes *all* possible groups of contiguous bins. In the second case, $\mathbf{L}$ contains a *proper subset*, which reduces the running time. In both cases, all v_i values in $\mathbf{L}$ are set to $1/\epsilon_3$, which is the expected error incurred by the Noise Addition submodule.

- *Absolute/squared error metric:* There are also two options for the error metric μ utilized by the Grouping submodule; absolute as in [2, 13] or squared as in [20, 22]. As shown later in the paper, this choice impacts both utility and performance.

Hierarchical scheme. The scheme captures data-oblivious methods. As shown in Figure 5, it consists of connectors C_1 and C_2, t Hierarchy Level modules (where t depends on the input public parameters), and a Query Processor. It receives as input a histogram $\mathbf{h}$, privacy budget ϵ, and public parameters $\mathbf{L}$ and f, where $\mathbf{L} = \mathbf{1}$ and f is the fan-out of the tree[3]. Connector C_1 initially receives $\mathbf{h}$, ϵ, $\mathbf{L}$ and f. Based on $\mathbf{h}$ and f, it creates an aggregate tree, and determines the tree height t. It next perceives each level of the tree as a histogram $\mathbf{h}_\ell$ for $\ell = 1, \ldots, t$. Finally, it splits the budget ϵ into t budgets ϵ/t and forwards $\mathbf{h}_\ell$ and $(\mathbf{1}, \epsilon/t, t, \ell)$ to the ℓ^{th} Hierarchy Level module.

The ℓ^{th} Hierarchy Level module sends a noisy histogram $\hat{\mathbf{h}}_\ell$ to C_2. The latter assembles a noisy tree $\hat{\mathbf{T}}$ from these histograms and forwards it to the Query Processor. In order to maximize utility, in our implentation the Query Processor answers range-sum queries by combining nodes from the noisy tree using the method of [10] (see Section 2.2).

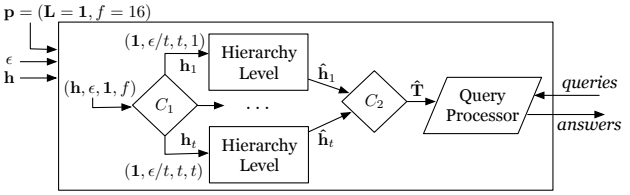


Figure 5: Hierarchical scheme

Each Hierarchy Level module ℓ offers $\alpha_\ell\epsilon/t$-differential privacy. Moreover, the modules work on non-disjoint sensitive inputs. As such, due to Theorem 3, the Hierarchical scheme offers $(\sum_{\ell=1}^{t} \frac{\alpha_\ell \epsilon}{t})$-differential privacy. Note that the ℓ^{th} Hierarchy Level module sets its α_ℓ as defined in [4] (through a closed formula based on t, ℓ), in a way that *guarantees* that $\sum_{\ell=1}^{t} \alpha_\ell = t$. Consequently, the Hierarchical scheme satisfies ϵ-differential privacy.

DAWA-like scheme. This scheme is a generalization of DAWA [13]. Recall that, in addition to a smoothing stage, DAWA employs the matrix mechanism, which receives as input all the possible range-sum queries. We abstract these two stages, so that any smoothing and fixed-queries scheme can be combined to realize DAWA's concept.

Figure 6 depicts the mechanics of the scheme. It consists of modules Smoothing and Fixed Queries, a connector, and

[3] In our implementation, we set $f = 16$ because it is optimal in terms of utility for range-sum queries [17].

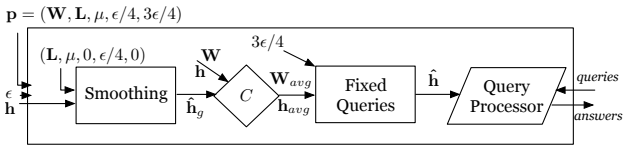


Figure 6: DAWA-like scheme

a Query Processor. It receives as input histogram $\mathbf{h}$, budget ϵ, and parameters $(\mathbf{W}, \mathbf{L}, \mu, \epsilon/4, 3\epsilon/4)$. Following [13], budget $\epsilon/4$ is allocated to the Smoothing module, and $3\epsilon/4$ to Fixed Queries. Vector $\mathbf{W}$ holds all possible $O(n^2)$ range-sum queries; μ defines the utilized error metric by the Smoothing; $\mathbf{L}$ contains the permissible groups. For each group g_i in $\mathbf{L}$, v_i is set to $4/(3\epsilon)$, which is the expected error due to the subsequent noise addition by the Fixed Queries module.

The Smoothing module takes as input $\mathbf{h}$ and parameters $(\mathbf{L}, \mu, 0, \epsilon/4, 0)$, and outputs a noisy histogram $\hat{\mathbf{h}}_g$ that incorporates the group labels. Connector C receives $\hat{\mathbf{h}}_g$, $\mathbf{W}$ and $\mathbf{h}$. It smooths $\mathbf{h}$ using $\hat{\mathbf{h}}_g$, and creates $\mathbf{h}_{avg}$. Then, it modifies workload $\mathbf{W}$ to $\mathbf{W}_{avg}$ to reflect the queries on $\mathbf{h}_{avg}$. The technical details of this conversion are included in [13]. Finally, it feeds $(\mathbf{h}_{avg}, \mathbf{W}_{avg})$ to Fixed Queries, which receives budget $3\epsilon/4$. This module computes and forwards a noisy histogram $\hat{\mathbf{h}}$ to the Query Processor, which answers range-sums by summing the bins included in the query range.

The Smoothing module satisfies $\epsilon/4$-differential privacy, while the Fixed Queries module $3\epsilon/4$-differential privacy. Both work on non-disjoint inputs and, therefore, the whole scheme satisfies ϵ-differential privacy.

4. GROUPING AND METRICS

The Grouping submodule of Smoothing determines the way the bins are privately grouped. In all existing schemes, this is modeled as an optimization problem where the resulting grouping must minimize a certain error metric. In this section, we first present in detail the two error metrics used in the literature, namely *absolute* [2, 13] and *squared* [20, 22] error, explain their usage, and analyze the overall time complexity of Grouping in each case. Next, we introduce an *optimal* way to compute the squared error, which (i) reduces the time complexity of the current best method by a factor of n, and (ii) improves the accuracy of Smoothing.

Recall that Grouping takes as input a histogram $\hat{\mathbf{h}}_o$, a privacy budget ϵ_2, public vector $\mathbf{L}$, and an error metric μ. Its goal is to find the groups that minimize μ, while satisfying ϵ_2-differential privacy. Let G be a *grouping strategy*, i.e., a set of $|G|$ groups of *contiguous* bins that cover all histogram bins and are mutually disjoint. Let b_j be a bin value, and $\bar{g}_i$ the average of the bins in group $g_i \in G$, i.e., $\bar{g}_i = \sum_{b_j \in g_i} b_j / |g_i|$.

The total error has two components. The first is due to the smoothing process and depends on the difference between the value b_j of a bin and the average $\bar{g}_i$ of the group in which it belongs. The second component is due to the noise injected by the module that succeeds grouping. For each group g_i, $\mathbf{L}$ contains a value v_i that corresponds to the latter. The *absolute* and *squared* error metrics combine the two components in different ways. Both metrics represent the collective error per bin, rather than the final error in a range-sum query. However, they are in general good indicators of accuracy and their minimization is likely to maximize utility.

Absolute error. This metric is defined in [2, 13] as:

$$err_1 = \sum_{i=1}^{|G|} \left(\sum_{b_j \in g_i} |b_j - \bar{g}_i| + v_i \right) \quad (1)$$

The state-of-the-art algorithm that uses the absolute error is the Smoothing module of DAWA [13], which works as follows. It first calculates the cost $c_i = \sum_{b_j \in g_i} |b_j - \bar{g}_i| + v_i$ of each group g_i in Equation 1 by utilizing a binary search tree in $O(\log n)$ time. Then, it adds noise with scale $1/(\epsilon_2 |g_i|)$ to c_i producing $\hat{c}_i = \sum_{b_j \in g_i} |b_j - \bar{g}_i| + v_i + Lap(1/(\epsilon_2 |g_i|))$. Finally, it finds the groups that minimize $\hat{err}_1 = \sum_{i=1}^{|G|} \hat{c}_i$ using dynamic programming in $O(n^2)$ time. The authors prove that an optimization algorithm that operates with such noisy costs ensures ϵ_2-differential privacy. The total time of Grouping is dominated by that of computing the costs of all the $O(n^2)$ groups, which is $O(n^2 \log n)$.

Squared error. This metric is defined in [20, 22] as:

$$err_2 = \sum_{i=1}^{|G|} \left(\sum_{b_j \in g_i} (b_j - \bar{g}_i)^2 + v_i^2 \right) \quad (2)$$

The state-of-the-art grouping algorithm that utilizes the squared error is AHP [22], which works as follows. It adds noise with scale $1/\epsilon_2$ to each bin of the initial histogram, and computes cost $\hat{c}_i = \sum_{\hat{b_j} \in g_i} (\hat{b_j} - \bar{g}_i)^2 + v_i^2$, where $\hat{b_j}$ is a noisy bin value, and $\bar{g}_i$ the average of a group of noisy bins. Finally, it finds the groups that minimize $\hat{err}_2 = \sum_{i=1}^{|G|} \hat{c}_i$. The algorithm satisfies ϵ_2-differential privacy because it operates on values perturbed with noise scale $1/\epsilon_2$. Its time complexity is $O(n^3)$.

The following theorem provides a lower bound on the time complexity of Grouping, in the case that all possible groups of contiguous bins are considered. The lower bound applies to *both* error metrics.

THEOREM 4. *A grouping algorithm on a histogram with n bins runs in $\Omega(n^2)$.*

PROOF. The number of all the possible groups is $\Theta(n^2)$. This is because we have n groups of size 1, $n-1$ groups of size 2, and so on (recall that a permissible group can only consist of contiguous bins). Thus, the total number of groups is $n + (n-1) + (n-2) + \ldots + 1 = \frac{n(n+1)}{2}$. It suffices to prove that there is an input for which any algorithm must check all the possible groups at least once.

We build a histogram such that every group g_i contributes cost $\hat{c}_i = |g_i|$ (i.e., equal to its cardinality) to the error metric. In this scenario, *any* grouping strategy G minimizes the error metric, since *every* G leads to error $\sum_{g_i \in G} \hat{c}_i = n$. Now suppose that we reduce the cost of a random group g_j to $(|g_j| - \delta)$ for some $\delta > 0$. Any grouping strategy that includes g_j will result in error $n - \delta$, whereas any other will result in n. Therefore, the grouping strategy G^* that minimizes the error metric *must* include g_j. Since g_j is a random group, the algorithm that finds G^* must check the $\hat{c}_i$ of *every* group g_i in order to find g_j. This concludes our proof. □

We next present an algorithm that minimizes the squared error $e\hat{r}r_2$ in $O(n^2)$. Therefore, due to the lower bound in Theorem 4, our algorithm is *optimal*. Given that $\hat{g}_i = \frac{\sum_{\hat{b}_j \in g_i} \hat{b}_j}{|g_i|}$, we observe that the cost of each group can be rewritten as follows.

$$\hat{c}_i = \sum_{\hat{b}_j \in g_i} \left(\hat{b_j} - \bar{g}_i\right)^2 + v_i^2 = \sum_{\hat{b}_j \in g_i} \hat{b_j}^2 - \frac{\left(\sum_{\hat{b}_j \in g_i} \hat{b}_j\right)^2}{|g_i|} + v_i^2$$

Based on the above equation, we can efficiently compute the cost of each group using the following procedure. Initially, we add noise with scale $1/\epsilon_2$ to every histogram bin. In a *pre-processing stage*, we build vector $\mathbf{v}_1$ that stores the noisy bin values $\hat{b}_j$, and vector $\mathbf{v}_2$ that stores their squares $\hat{b}_j^2$. Subsequently, we construct the *prefix sums* for each vector. Specifically, the prefix sums for $\mathbf{v}_1$ ($\mathbf{v}_2$) is a vector $\mathbf{v}'_1$ ($\mathbf{v}'_2$), such that $\mathbf{v}'_1[j] = \sum_{i=1}^{j} \mathbf{v}_1[i]$ ($\mathbf{v}'_2[j] = \sum_{i=1}^{j} \mathbf{v}_2[i]$). The pre-processing takes $O(n)$ time. For each group g_i over contiguous bins $l, l+1, \ldots, u$, we can compute $\sum_{\hat{b}_j \in g_i} \hat{b}_j$ as $\mathbf{v}'_1[u] - \mathbf{v}'_1[l-1]$ and $\sum_{\hat{b}_j \in g_i} \hat{b}_j^2$ as $\mathbf{v}'_2[u] - \mathbf{v}'_2[l-1]$ in $O(1)$ time. Thus, the cost of any group requires $O(1)$ time. Since there are $O(n^2)$ possible groups, we can calculate all their costs in $O(n^2)$. Finally, in order to find the grouping strategy that minimizes $e\hat{r}r_2$, we employ the dynamic programming procedure of [13], which runs in $O(n^2)$ time. Therefore, our algorithm has total running time $O(n^2)$.

We conclude this section with an improvement on the accuracy yielded by the use of the squared error. Recall that our algorithm computes the group costs on the noisy histogram in order to ensure ϵ_2-differential privacy. Thus, the grouping strategy that minimizes $e\hat{r}r_2$, may not minimize err_2 (defined on the original bins). In order to alleviate the effects of the extra noise in $e\hat{r}r_2$ we exploit the following observation. Using a similar approach as in the proof of Lemma 1 in [20], we can show that each group is expected to have its cost increased due to noise by $2\frac{|g_i|-1}{\epsilon_2^2}$, i.e., proportionally to the group size. The additional error leads to smaller groups for $e\hat{r}r_2$ minimization, compared to err_2. To mitigate this, we reduce the calculated cost $\hat{c}_i$ of each group g_i by $2\frac{|g_i|-1}{\epsilon_2^2}$, before feeding it to the dynamic programming procedure. Compared to its direct competitor AHP [22], our algorithm improves the utility by up to 70% and the complexity by n.

5. NOVEL SCHEMES

We design two schemes based on our modular framework. The first, called *Subtree Smoothing*, constitutes the first approach that seamlessly combines smoothing with aggregate trees, running in $O(n)$ time. The second, called *Smoothed Prefix Sums*, reduces the time complexity of DAWA by a factor of n, while maintaining its utility.

5.1 Subtree Smoothing Scheme

Recall that the Hierarchical scheme builds an aggregate tree in order to compose the range-sum answer from a small number of noisy values, thus reducing the error resulting from noise aggregation as opposed to LPA. However, due to the publication of multiple non-disjoint histograms (one per level), it must add more noise per level than LPA. On the other hand, the Smoothing scheme reduces the sensitivity of a set of bins via grouping and averaging, thus lowering the required noise. Our Subtree Smoothing scheme builds an aggregate tree similar to the Hierarchical scheme (thus reducing the error from noise aggregation), but *smooths entire subtrees* via grouping and averaging similar to the Smoothing scheme (thus reducing the per-level, per-bin noise).

Figure 7 illustrates the main idea. The scheme runs the Smoothing module *only once* for the leaf level (i.e., for $\mathbf{h}$), setting as permissible groups only those that correspond to the leaves of *full* subtrees. Suppose that the black nodes in the figure comprise a group in the returned grouping strategy. We refer to the root of the subtree corresponding to a group as the *group root*. Next, the scheme creates the aggregate tree, *pruning* the nodes under the group roots (black nodes). Subsequently, it feeds each level of this aggregate tree to a Hierarchy Level module, which outputs a noisy histogram. The final noisy histograms comprise a noisy tree. Finally, the scheme puts the pruned nodes back to the tree, deriving their values from their corresponding group root. Specifically, the value in the group root is distributed evenly across the nodes of the same level in the subtree. This is equivalent to smoothing the nodes at each level of the subtree via averaging.

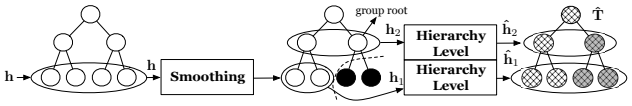


Figure 7: Subtree smoothing example

Figure 8 illustrates the modules of the Subtree Smoothing scheme. There is a single Smoothing module, t Hierarchy Level modules, two connectors, and a Query Processor. The input of the scheme includes the sensitive histogram $\mathbf{h}$, privacy budget ϵ, and public parameters $\mathbf{p} = (\mathbf{L}, \mu, \epsilon/4, 3\epsilon/4, f)$. $\mathbf{L}$ is the set of permissible groups for the Smoothing module and their associated v_i values; μ is the error metric; $\epsilon/4$ is the budget allocated to the Smoothing module; $3\epsilon/4$ is the budget distributed evenly to the t Hierarchy Level modules; f is the fan-out of the aggregate tree, and t is its derived height. Following [17], we set $f = 16$.

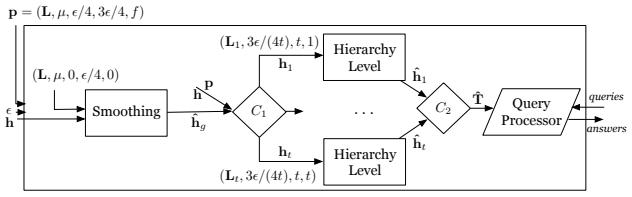


Figure 8: Subtree Smoothing scheme

The Smoothing module takes as input $\mathbf{h}$ and parameters $(\mathbf{L}, \mu, 0, \epsilon/4, 0)$, and outputs a noisy histogram $\hat{\mathbf{h}}_g$. $\mathbf{L}$ contains the groups formed by bins that can be leaves of full subtrees in the final aggregate tree. Their v_i values are all set to $4t/(3\epsilon)$. Ordering must *always* be deactivated because the final aggregate tree is built considering the order of the bins in $\mathbf{h}$. If Ordering were activated, Grouping could select a group g_i, whose bins are not the leaves of a full subtree on $\mathbf{h}$ (since Ordering may permute the bins of $\mathbf{h}$). Therefore, g_i could not determine a group root to smooth a subtree,

thus violating the scheme. The Grouping submodule takes budget $\epsilon/4$. The Noise Addition submodule receives 0 budget; the returned $\hat{\mathbf{h}}_g$ contains only the grouping information, which is used later.

Connector C_1 receives $\mathbf{h}$, $\hat{\mathbf{h}}_g$, budget $3\epsilon/4$ and fan-out f. It builds the aggregate tree utilizing $\mathbf{h}$, $\hat{\mathbf{h}}_g$, f, and considers each tree level ℓ as a histogram $\mathbf{h}_\ell$ to be sent to the ℓ^{th} Hierarchy Level module. For every pruned node j in the aggregate tree (i.e., black node in Figure 7) at level ℓ, it sets its scalar to $\mathbf{L}_\ell[j] = 0$, and for any other node j' it sets $\mathbf{L}_\ell[j'] = 1$.

The ℓ^{th} Hierarchy Level module receives the histogram $\mathbf{h}_\ell$. For each bin b_j in $\mathbf{h}_\ell$, it adds noise with scale $\frac{4t}{\alpha_\ell \cdot 3\epsilon \cdot \mathbf{L}_\ell[j]}$. If b_j is a node to be pruned, then $\mathbf{L}_\ell[j] = 0$, and b_j is perturbed with infinite noise, while a special annotation is added to its label. This is essentially equivalent to completely disregarding the pruned nodes. Otherwise, $\mathbf{L}_\ell[j] = 1$ and the module adds noise with scale $\frac{4t}{\alpha_\ell \cdot 3\epsilon}$. This procedure ensures that each Hierarchy Level module satisfies $\frac{\alpha_\ell \cdot 3\epsilon}{4t}$-differential privacy (by a direct application of Theorems 1 and 2).

Connector C_2 receives the noisy histograms from the Hierarchy Level modules. First, it assembles a noisy aggregate tree from the histograms. Next, it substitutes the values of the nodes that received infinite noise in the Hierarchy Level modules, with the values derived from their group root, as we explained in the context of Figure 7. Finally, it forwards the resulting tree $\hat{\mathbf{T}}$ to the Query Processor, which answers range-sum queries using the technique of [10].

We next analyze the privacy of the scheme. The Smoothing module spends budget $\epsilon/4$ and satisfies $\epsilon/4$-differential privacy. Each of the t Hierarchy Level modules satisfies $\frac{\alpha_\ell \cdot 3\epsilon}{4t}$-differential privacy as explained above. Moreover, all the modules work on non-disjoint inputs. Due to Theorem 3, and recalling that the α_ℓ values are selected according to [4] such that $\sum_{i=1}^{t} \alpha_\ell = t$, the whole scheme satisfies ϵ-differential privacy.

Finally, the running time of the scheme depends on the error metric μ. Observe that the number of groups examined by Grouping is equal to the number of nodes in the aggregate tree, i.e., $O(n)$. For the case when μ is the absolute error, the running time of Grouping is $O(n \log n)$, using the smoothing algorithm of [13]. If μ is the squared error metric, the complexity is $O(n)$ using our optimal algorithm from Section 4. Each Hierarchy Level module runs in time linear in the number of input nodes, thus, all the t Hierarchy Level modules run collectively in $O(n)$. In our experiments, we demonstrate that the error metric in Subtree Smoothing does not significantly affect the utility. Hence, we fix μ to the more efficient square error, which yields total running time $O(n)$.

Utility Optimization. Instead of completely disregarding the nodes of a pruned subtree, we can actually utilize them to reduce the noise of its root. Specifically, for each level of the pruned subtree, we sum the node values and add noise, producing a noisy estimation of the root. Subsequently, we use the *average* of these estimations as the root noisy value. The mechanism then proceeds as described above, i.e., the root value is distributed evenly among the subtree nodes. This reduces the *squared* error of the root value by t', where t' is the height of the subtree. We omit the proof due to its simplicity.

5.2 Smoothed Prefix Sums Scheme

This scheme is based on *prefix sums* [11]. A prefix sum query over $\mathbf{h}$ is simply described by an index j, and returns the sum of bins $b_1, \ldots, b_j$, i.e., $\sum_{i=1}^{j} \mathbf{h}[i]$. There are n prefix sums, hereafter represented by a vector $\mathbf{s}$ such that $\mathbf{s}[j] = \sum_{i=1}^{j} \mathbf{h}[i]$ for $j = 1, \ldots, n$. Moreover, observe that *any arbitrary* range-sum query can be always computed by the subtraction of *exactly two* prefix sums; for instance, range-sum $[i_l, i_u]$ is answered as $\mathbf{s}[i_u] - \mathbf{s}[i_l - 1]$.

The Smoothed Prefix Sum scheme takes advantage of the fact that there are n prefix sums, as opposed to $O(n^2)$ possible range queries, to improve the complexity of DAWA-like methods by a factor of n. It considers the prefix sums as the fixed workload $\mathbf{W}$, and produces a noisy histogram $\hat{\mathbf{h}}$. The latter enables the computation of a vector of noisy prefix sums $\hat{\mathbf{s}}$, such that $\hat{\mathbf{s}}[i] = \sum_{i=1}^{j} \hat{\mathbf{h}}[i]$. Vector $\hat{\mathbf{s}}$ is fed to the Query Processor, which computes in $O(1)$ time any range-sum $[i_l, i_u]$ as $\hat{\mathbf{s}}[i_u] - \hat{\mathbf{s}}[i_l - 1]$. Since the Fixed Queries module leads to highly accurate $\hat{\mathbf{s}}[i]$, the range-sum result is expected to have very low error.

Figure 9 depicts the Smoothed Prefix Sums scheme. The only differences with respect to the DAWA-like scheme of Figure 6 are (i) the workload $\mathbf{W}$, which now contains the prefix sums, and (ii) an extra connector C_2 before the Query Processor, which converts the output histogram $\hat{\mathbf{h}}$ into a prefix sums array $\hat{\mathbf{s}}$.

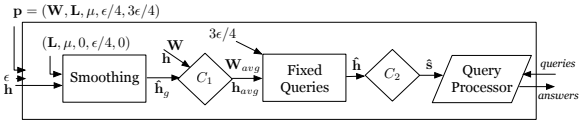


Figure 9: Smoothed Prefix Sums scheme

The Smoothing module satisfies $\epsilon/4$-differential privacy, while the Fixed Queries module $3\epsilon/4$-differential privacy. Both work on non-disjoint inputs and, therefore, the whole scheme satisfies ϵ-differential privacy. Its time complexity is $O(|\mathbf{W}| n \log n) = O(n^2 \log n)$, since now $|\mathbf{W}| = n$. The expected error is at most two times larger than that of DAWA because Smoothed Prefix Sums subtracts two noisy values from the prefix sums array to answer a range query, while DAWA essentially returns a value for the same range. However, in our experiments we demonstrate that the utility of Smoothed Prefix Sums is practically the same as that of the DAWA scheme.

A remark concerns allocation of budget ϵ to the various modules. In this work, we followed the empirical allocation policies of the existing schemes. Determining the optimal allocation is out of our scope, but we consider it as an interesting problem for future work. Finally, except for LPA and the Hierarchical scheme, where the expected error is expressed theoretically, the rest of the schemes are highly *data-dependent*. Therefore, their utility must be experimentally evaluated under different real settings, a task we undertake in the next section.

6. EXPERIMENTAL EVALUATION

In this section we evaluate the methods of Table 1 in terms of utility and efficiency. LPA corresponds to the Laplace Perturbation Algorithm. H implements the Hierarchical scheme,

Table 1: Summary of schemes

Scheme	Abbrv	Time
Laplace perturbation algorithm	LPA	$O(n)$
Hierarchical scheme	H	$O(n)$
Smoothing with absolute error metric	S_1	$O(n^2 \log n)$
Approximate Smoothing	$\tilde{\mathsf{S}}$	$O(n \log^2 n)$
Smoothing with squared error metric	S_2	$O(n^2)$
Smoothing with ordering	S_o	$O(n^2)$
DAWA	DAWA	$O(n^3 \log n)$
Subtree smoothing	SUB	$O(n)$
Smoothed prefix sums	SPS	$O(n^2 \log n)$

using all optimizations of [17]. S_1 incorporates the smoothing algorithm of DAWA [13], based on the absolute error metric. $\tilde{\mathsf{S}}$ is the approximate version of S_1 that considers only a subset of the possible groups [13]. S_2 applies smoothing using the squared error metric, and it utilizes the quadratic algorithm and utility optimization described in Section 4. S_o orders the bin values [22], and then uses S_2 for smoothing. DAWA [13] is implemented with input all the possible range queries. SUB is our Subtree Smoothing scheme based on the squared error metric as it offers similar utility to the absolute metric, and is faster by a $\log n$ factor. It also contains the utility optimization technique described at the end of Section 5.1. SPS is our Smoothed Prefix Sum scheme. For both SPS and DAWA, we use the absolute error metric, but the choice of metric does not affect either their utility, or performance.

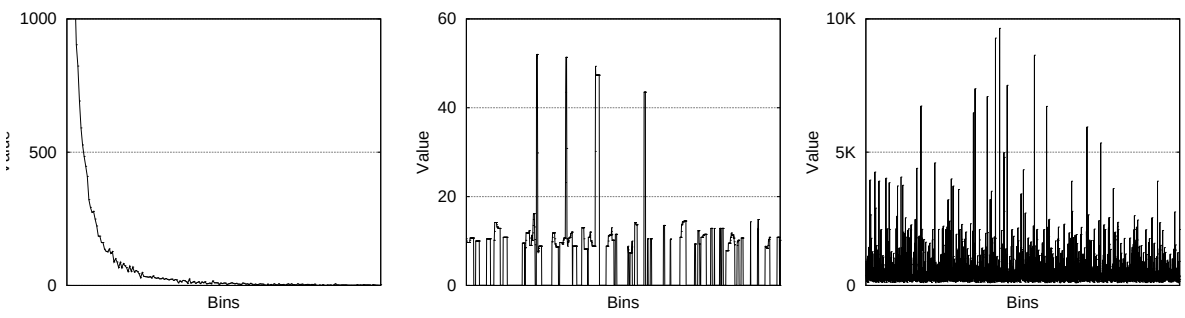
Our evaluation includes all the dominant techniques in their respective settings. Specifically, the smoothing module of DAWA (S_1) offers better utility and time complexity than previous methods that are based on the absolute error metric and check all possible groupings [13]. Its approximate counterpart $\tilde{\mathsf{S}}$ also dominates its competitor P-HP, which in turn has been shown to outperform EFPA and SF [2] (details about these methods can be found in Section 2.2). Among the exhaustive techniques based on the squared error, the state-of-the-art is AHP [22], which is dominated by S_2 in terms of running time and utility, as explained in Section 4. Moreover, the approximate methods using the squared error metric have the same quadratic complexity as S_2, while at best they can reach the same utility. Finally, the optimizations incorporated in H have been shown to yield the best hierarchical method in the survey of [17].

We implemented all methods of Table 1 in Java, and conducted experiments on an Intel Core i5 CPU 2.53GHz with 4GB RAM, running Windows 7. Following the literature, we assess utility using the *Mean Squared Error* (MSE), fixing $\epsilon = 1$. The cardinality of the range-sum queries varies between 10% and 50% of the input histogram size. Every reported result is the average of 100 executions, each containing 2000 random queries of the selected cardinality.

We used three real datasets, henceforth referred to as *Citations* [8], *Rome* [1], and *GoWalla* [3]. In *Citations*, we created a histogram of 2414 bins as in [17], where each bin b_i is the number of papers cited i times. A range-sum query $[i_l, i_u]$ returns the papers cited between i_l and i_u times. The *Rome* dataset consists of 14420 bins, where each bin b_i is the number of cars on a specific road at time instance i. A range-sum query asks for the traffic at this road segment during a time interval. Finally, *GoWalla* consists of user check-ins at 2791 locations. We sorted the locations in ascending order of their x-coordinates as in [17], and viewed them as histogram bins. A range-sum query returns the number of users in a vertical geographical strip.

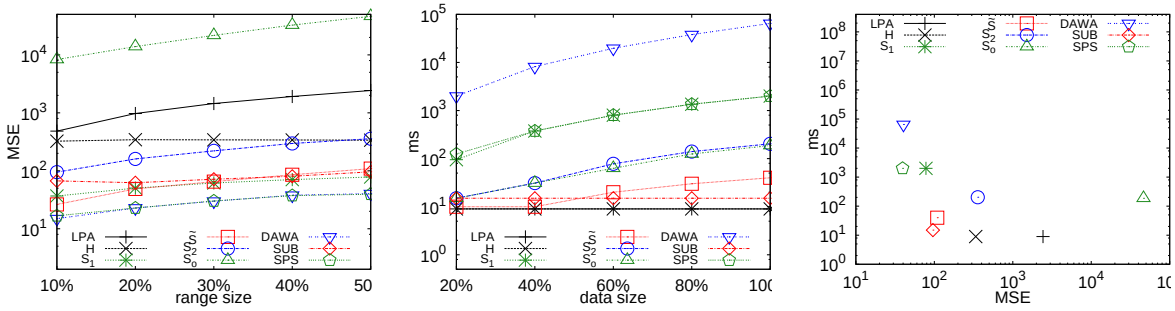
Citations, *Rome*, and *GoWalla* feature considerably different distributions, depicted in Figures 10(a), 10(b), and 10(c), respectively. *Citations* is very sparse, and its consecutive bin values are similar, especially for bins that correspond to numerous citations (most such bins have 0 values). *Rome* exhibits high fluctuations at specific contiguous bins (reflecting peak hours), and includes numerous small values (reflecting non-peak hours). Finally, *GoWalla* contains almost random values, since the number of check-ins is independent of the value of the x-coordinate.

Figure 11(a) plots the MSE for *Citations*, when varying the range size (expressed as a fraction of the number of bins). SPS and DAWA achieve the highest accuracy. The error of $\mathsf{S}_1, \tilde{\mathsf{S}}$, and SUB is up to two times higher, while that of $\mathsf{H}, \mathsf{LPA}, \mathsf{S}_2$ is more than an order of magnitude larger. S_o exhibits the worst performance because the noise injected by ordering yields a poor grouping strategy. The low MSE of SPS and DAWA is mainly due to their effective combination of smoothing and the matrix mechanism. Their almost identical error confirms our claim in Section 5.2 that feeding prefix sums to the matrix mechanism of the Fixed Queries module (SPS) leads to the same practical utility as providing all the possible ranges (DAWA). In general, all smoothing techniques perform well because consecutive bins have similar values, leading to groups with low error (this dataset yields a small number of large groups). This also explains the marginal difference of S_1 and $\tilde{\mathsf{S}}$; $\tilde{\mathsf{S}}$ can easily find a good grouping strategy even though it does not explore all possible groups. In contrast, S_2 performs worse than S_1 and $\tilde{\mathsf{S}}$, as the squared error metric is sensitive to some small fluctuations in the dataset, which leads to unnecessarily small groups. Methods that do not rely on aggregate trees (i.e., LPA, $\mathsf{S}_1, \tilde{\mathsf{S}}$, S_2 and S_o) are affected by the range size, as the



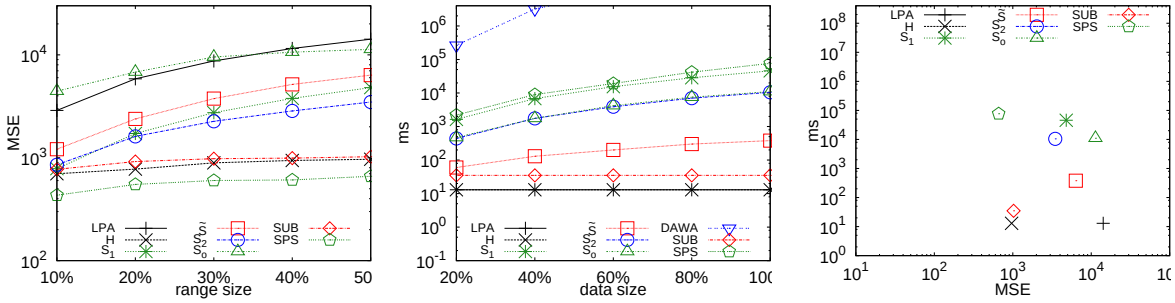
(a) *Citations* (b) *Rome* (c) *GoWalla*

Figure 10: Data distribution



(a) *MSE* (b) *Running Time* (c) *Running Time vs. Error*

Figure 11: *Citations*



(a) *MSE* (b) *Running Time* (c) *Running Time vs. Error*

Figure 12: *Rome*

number of noisy values participating in the calculation of the range-sum (and, thus, the resulting error accumulation) increases linearly with the size. On the other hand, the range has small effect on the utility of hierarchical methods, which increases logarithmically with the range size.

Figure 11(b) evaluates the CPU-time as a function of the data size. In order to reduce the data size to a percentage x%, we select the first x% values and the corresponding bins. H and LPA are the fastest methods as expected by their linear complexity. SUB is slightly more expensive because of the additional smoothing step at the leaf level of the aggregate tree. The next method in terms of efficiency is $\tilde{S}$, with complexity $O(n \log^2 n)$, followed by the quadratic S_2 and S_o. S_1 and SPS have almost the same running time due to their identical complexity $O(n^2 \log n)$. DAWA ($O(n^3 \log n)$) is more than an order of magnitude more expensive than any other method.

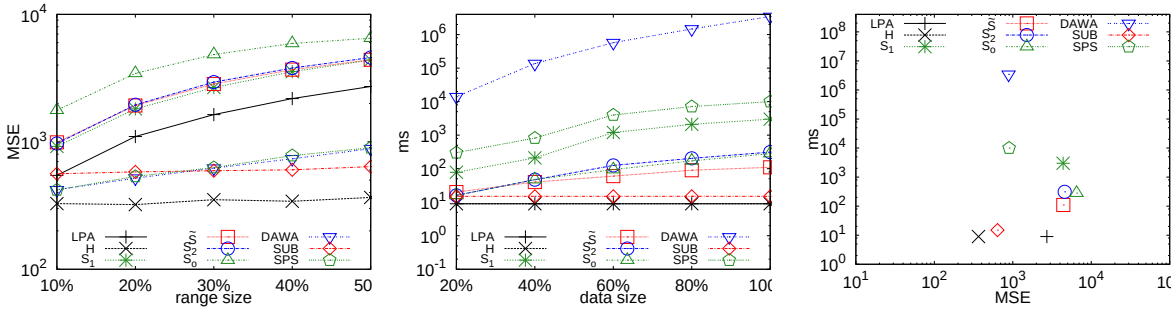
In order to demonstrate the utility-efficiency trade-off, Figure 11(c) plots the error (x-axis) versus time (y-axis), when fixing the range-sum size to 30% of the bins and using the entire dataset. The best solutions on both aspects lie closest to the axes origin. Although SPS and DAWA feature the best utility, they are also the most expensive. However, DAWA is dominated by SPS, which is much more efficient. On the other hand, fast methods such as LPA and H incur high error. In between the two extremes lies SUB, which is almost as fast as H and LPA, but exhibits 3.5 times lower error than H and an order of magnitude lower than LPA.

Figure 12(a) assesses the utility of the schemes on the *Rome* dataset. The results for DAWA are omitted, since it failed to terminate within a reasonable time (in fact, we estimated that it would take approximately three months to finish for this dataset). Similar to Figure 11(a), SPS is the best scheme, reducing the error of the next best solutions (H and SUB) by up to 70%. H and SUB have almost identical error. Compared to Figure 11(a), smoothing-based techniques have inferior performance because, due to the high fluctuations of the dataset, it is difficult to find effective grouping strategies. As opposed to *Citations*, *Rome* yields a large number of small groups. S_1, S_2 and $\tilde{S}$ achieve gains through smoothing only for small ranges. S_2 outperforms S_1 and $\tilde{S}$ because the squared error distinguishes small fluctuations, whereas the absolute error erroneously merges small groups into larger ones. Similarly, $\tilde{S}$ results in up to 50% worse error than S_1 and up to an order of magnitude worse than S_2 because the arbitrary set of the examined groups of the former does not include the groups that minimize the error. Finally, S_o and LPA are the worst techniques.

Figure 12(b) measures the CPU-time versus the data size. The results and the relative order of the schemes are consistent with Figure 11(b). DAWA can only run for up to 40% of the dataset. Figure 12(c) plots the error versus efficiency for *Rome*. Again SPS and LPA lie at the two extremes of utility and efficiency, respectively. SUB and H provide the best trade-off, dominating all the schemes but SPS.

Figure 13(a) depicts the MSE on *GoWalla* as a function of the range size. *GoWalla* features almost random bin values. Consequently, smoothing spends privacy budget, without finding groups that lead to noise reduction. The schemes that depend solely on smoothing, i.e., S_1, S_2, S_o and $\tilde{S}$, are outperformed even by LPA. On the other hand, SUB, DAWA and SPS are more robust to the dataset characteristics, since they inherit the benefits of the aggregate tree and matrix mechanism, respectively. For this dataset, a simple aggregate tree generated by H is the best method.

Figure 13(b) plots the CPU-time. The relative performance is similar to the previous diagrams. DAWA terminates because *GoWalla* is smaller than *Rome* (2791 versus



(a) *MSE* (b) *Running Time* (c) *Running Time vs. Error*

Figure 13: *GoWalla*

14420 bins). Figure 13(c) shows the error-efficiency tradeoff for *GoWalla*. H dominates all solutions in both aspects, whereas SUB lies close to H.

Summary. Our experiments demonstrate that data-aware techniques lead to considerable error reductions for datasets that have similar values in consecutive bins. However, the gains of smoothing vanish in datasets with numerous high fluctuations. In these scenarios, data-oblivious methods are preferable because they do not waste privacy budget on smoothing. In between these two extremes lie schemes that integrate smoothing with other modules (i.e., SUB and SPS), and are more robust to the dataset characteristics. Specifically, SPS performed identically to DAWA in terms of utility, while reducing the complexity by a factor of n to $O(n^2 \log n)$. On the other hand, for time-critical applications (e.g., real-time traffic), where even SPS may be too slow, SUB achieves comparable accuracy, while having the lowest running time ($O(n)$) among all the data-aware methods.

7. CONCLUSION

This paper introduces a modular framework for differentially private histogram publication. We first express existing methods in the framework, and identify opportunities for optimization. We then design a new optimal algorithm for smoothing, which improves utility and reduces the running time of the current state-of-the-art. Next, we develop new schemes that combine heterogeneous privacy techniques, previously deemed unrelated. Finally, we experiment on three datasets with diverse characteristics. Our results confirm that our modular approach enables the design of schemes that (i) are tailored to the data characteristics of the application at hand, and (ii) offer a desirable tradeoff between efficiency and utility.

8. REFERENCES

[1] `http://apprendistato.informaservizi.it/`.

[2] G. Acs, C. Castelluccia, and R. Chen. Differentially private histogram publishing through lossy compression. In *ICDM*, 2012.

[3] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: User movement in location-based social networks. In *SIGKDD*, 2011.

[4] G. Cormode, C. Procopiuc, D. Srivastava, E. Shen, and T. Yu. Differentially private spatial decompositions. In *ICDE*, 2012.

[5] C. Dwork. Differential privacy. In *ICALP*, 2006.

[6] C. Dwork. A firm foundation for private data analysis. *CACM*, 54(1):86–95, 2011.

[7] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.

[8] J. Gehrke, P. Ginsparg, and J. M. Kleinberg. Overview of the 2003 KDD cup. *SIGKDD Explorations*, 5(2):149–151, 2003.

[9] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. In *NIPS*, 2012.

[10] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. In *VLDB*, 2010.

[11] C.-T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in olap data cubes. In *SIGMOD*, 1997.

[12] G. Kellaris and S. Papadopoulos. Practical differential privacy via grouping and smoothing. In *VLDB*, 2013.

[13] C. Li, M. Hay, G. Miklau, and Y. Wang. A data-and workload-aware algorithm for range queries under differential privacy. *VLDB*, 2014.

[14] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *PODS*, 2010.

[15] F. McSherry and K. Talwar. Mechanism design via differential privacy. FOCS, 2007.

[16] F. D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. SIGMOD, 2009.

[17] W. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. In *VLDB*, 2013.

[18] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD*, 2010.

[19] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *TKDE*, 23(8):1200–1214, 2011.

[20] J. Xu, Z. Zhang, X. Xiao, Y. Yang, and Y. Ge. Differentially private histogram publication. In *ICDE*, 2012.

[21] G. Yuan, Z. Zhang, M. Winslett, X. Xiao, Y. Yang, and Z. Hao. LowRank mechanism: Optimizing batch queries under differential privacy. *VLDB*, 2012.

[22] X. Zhang, R. Chen, J. Xu, X. Meng, and Y. Xie. Towards accurate histogram publication under differential privacy. In *SDM*, 2014.